FAST, THREE-DIMENSIONAL, COLLISION-FREE MOTION PLANNING

Martin Herman

Reprinted from PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, San Francisco, California, April 7-10, 1986

IEEE COMPUTER SOCIETY
1730 Massachusetts Avenue, N.W.
Washington, D.C. 20036-1903

IEEE
COMPUTER
SOCIETY
PRESS

Fast, Three-Dimensional, Collision-Free Motion Planning

Martin Herman

Robot Systems Division
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

Issues dealing with fast, 3-D, collision-free motion planning are discussed, and a fast path planning system under development at NBS is described. The components of a general motion planner are outlined, and some of their computational aspects are discussed. It is argued that an octree representation of the obstacles in the world leads to fast path planning algorithms. The system we are developing uses such an octree representation. The robot and its swept-volume paths are approximated by primitive shapes so as to result in fast collision detection algorithms. The search for a path is performed in the octree space, and combines hypothesize and test, hill climbing, and A* .

1. Introduction

In order to develop robots that can operate in a wide variety of situations, fast, collision-free motion planning algorithms are necessary. Most previous planning algorithms have operated in two dimensions, often for mobile robot applications. Further, many of these assume that rotation of the robot can be ignored, and consider translation only. The computational expense of these algorithms would probably increase dramatically if applied to the extra degrees of freedom in three-dimensional motion. Of the planning algorithms developed for 3-D motion, many are very slow.

This paper discusses some of the issues related to fast 3-D motion planning, and presents such a system being developed at NBS. The system performs the path search in an octree space, and uses a hybrid search technique that combines hypothesize and test, hill climbing, and *A**.

2. *Computational Aspects of Motion Planning*

This section describes some major components of a general motion planning system, and discusses their computational aspects in the context of various task domains.

A program that automatically generates collision-free motions for robots requires the following major components:

1. World representation. A representation of the robot and its world is required. This representation accumulates the description of the world from outside sources. Common forms for such a representation are surface-based CAD models [Baer et al. 79, Requicha 80], swept volumes [Brooks 81], cellular arrays [Srihari 81], octrees [Meagher 82, Jackins & Tanimoto 80], and analytic surface equations.
2. World description acquisition. The world description may be obtained from various sources. These include manual input, a priori object data bases, sensory recognition modules, and sensory description modules.
3. Search space representation. The search for collision-free paths occurs in a search space. The representation might be the same as the world representation, but is often different. Examples of search space representations that are usually different from world representations are configuration spaces [Lozano-Perez 81], Voronoi-based spaces [Canny 85], generalized cylinder free spaces [Brooks 83], and medial axis free spaces [Ruff & Ahuja 84].
4. World-to-search-space mapping. When the search space representation is different from the world representation, a procedure that maps the world to the search space is required.
5. High-level task planner. In this component, the overall task of the robot is planned. Information about specific position constraints on the robot are sent to the path planner, which finds a collision-free path satisfying these constraints.
6. Path planner. This component uses a set of search techniques to find collision-free paths in the search space.
7. Trajectory planner. This component converts the path obtained by the search process into a trajectory that can be executed by the robot. Typically, the path planning process is concerned only with collision-free configurations in space, but not with velocity, acceleration, smoothness of motion, etc. These factors are handled by the trajectory planner. The output of this component forms the motion commands to the servo mechanisms of the robot.

Although these seven components have been described separately, their tasks often overlap and there may be no absolute demarcation between some of them.

Of these components, we will discuss three that often involve major computational expense: (1) acquiring the world description, (2) mapping the world description into the search space description, and (3) performing the search for a path. Let us discuss the computational expense in these three areas by categorizing task domains in three ways: (1) the number of times a path will be executed once it is found, (2) the amount of a priori knowledge available about the robot's world, and

Identification of commercial equipment in this paper is only for adequate description of our work. It does not imply recommendation by the National Bureau of Standards, nor that this equipment was necessarily the best available for the purpose.

(3) whether the world is basically static or dynamic.

Number of times a path will be executed. Industrial parts can be produced in factories either in large or in small quantities. When produced in large quantities, single motions might be performed thousands of times. Therefore, the time involved in generating trajectories for manipulators is not of great significance, since this is done offline only once for each task. The most important consideration is obtaining optimal paths that are efficient in terms of time of execution, robot wear and tear, etc. [Kambhampati & Davis 85, Thorpe 84].

When parts are produced in small quantities, single motions are performed a relatively small number of times. The time involved in generating trajectories might then become more important, and should be considered when choosing a motion planning technique.

Incomplete vs. complete a priori knowledge of the world. The following kinds of knowledge about the robot's world are relevant to path planning: (1) descriptions of the objects that lie in the world, (2) the positions and orientations of these objects at some point in time, and (3) the motions of these objects as a function of time. The degree of incompleteness in the knowledge of the world affects which strategies the robot should use in interacting with the world. If the incompleteness is in the form of small uncertainty in poses of objects, the world is said to be *uncertain*. This occurs frequently in factory tasks. In such cases, trajectories are commonly generated offline and executed by an intelligent program which uses force control, compliance, and simple vision to make minor modifications to the trajectories [Brady et al. 82].

If the incompleteness is in the form of unknown objects in the world or totally unknown poses of objects, the world is said to be *unstructured*. In such cases, a large portion of the trajectories may have to be generated online, after a description of the world has been obtained using complex sensory processing for recognition, description, and localization. At this point, of course, the time involved in generating the trajectory becomes very important. Let us consider this expense in terms of the three areas described previously. First, there is the expense of acquiring the world description using sensory processing and interpretation algorithms. This is a very large topic and will not be discussed here (see [Hong & Shneier 85a, Herman 85, Kent et al. 85, Besl and Jain 85]. Second, the expense of mapping the world description into a search space description may be significant, particularly since the output representations of sensory interpretation algorithms are often different from those used by path search algorithms. Third, the search itself should be very fast.

Static vs. dynamic world. A dynamic world is one that changes over time. Changes in the world may be due to (1) motion of the robot, (2) motion of objects, caused by the robot, (3) motion of objects, not caused by the robot, or (4) the appearance or disappearance of objects from the world, not caused by the robot (here the motion trajectories of the objects are not known by the robot).

In order to find and maintain collision-free trajectories in a dynamic world, the search space must be updated as changes occur. If the world changes in a predictable way, updates to the search space can be computed offline in advance; in fact the whole trajectory can be computed in advance. However, if the world changes in an unpredictable way, then it is unstructured, and the previous discussion about unstructured worlds applies. In addition, updating of the search space should occur incrementally, since it would probably be very expensive to regenerate the whole search space when only a small part of it has changed. Incremental updates are also necessary for static unstructured worlds where sensory information is obtained incrementally and trajectories must be generated while parts of the world are still unknown. Of course, the expense involved in performing the search in the updated search space is very important since this must be done online.

3. Fast Path Planning

In this paper, we consider some issues related to fast path planning in an unstructured, dynamic environment. The term "fast" is used informally here to specify a practical time frame in which the robot can plan and execute motions. Generally, this time frame might be on the order of a few minutes. Since our focus will be on fast path planning, many of the ideas will also relate to planning in structured, static environments, such as planning paths for factory manipulators involved in producing parts in small quantities.

Many of the approaches attempted previously are not adequate for this domain. The configuration space approach, for example, is computationally very expensive. It requires, first, mapping a world description into a configuration space, i.e., generating the configuration space obstacles [Lozano-Perez 81]. In general, this step is very time consuming. Further, it is not clear that this can be done incrementally. Second, the search must be performed in a high-dimensional space. Although the technique of slice projections [Lozano-Perez 81] reduces the computation involved, the explicit representation of the high-dimensional space can consume a large amount of memory. Searching a high-dimensional space can also be very time consuming. However most approaches share this problem, whether they represent the space explicitly or implicitly.

Several approaches for explicitly representing free space, such as with generalized cylinders [Brooks 83], medial axis transforms [Ruff & Ahuja 84], and Voronoi methods [Canny 85], are not adequate for this domain because of the computational expense involved in mapping a world description into the free space description. The free space description is also very sensitive to object motion.

The potential field approaches [Khatib 85, Buckley & Leifer 85] offer excellent possibilities for very fast obstacle avoidance. However, they suffer from being "too dumb"; they often get stuck at local minima in the potential field. These methods will have to be augmented with smarter, and probably more computationally expensive, algorithms to serve as general path planners.

4. Path Planning in Octree Space

In this paper, we will argue that an octree representation of the world leads to fast path planning. An octree [Meagher 82, Jackins & Tanimoto 80] is a recursive decomposition of a cubic space into subcubes. Initially, the whole space is represented by a single node in the tree, called the root node. If the cubic volume is homogenous (completely filled by an object or completely empty), then the root is not decomposed at all, and comprises the complete description of the space. Otherwise, it is split into eight equal subcubes (octants), which become the children of the root. This process continues until all the nodes are homogeneous, or until some resolution limit is

reached. Nodes corresponding to cubic regions that are completely full are called FULL leaf nodes. Nodes corresponding to empty regions are called EMPTY leaf nodes, and nodes corresponding to mixed regions (non-leaf nodes) are called MIXED nodes.

The techniques for path planning described here assume that octrees are used to represent Cartesian 3-space, and that path planning occurs in this space. Octrees have also been used to represent other search spaces [Faverjon 84]. Also, other hierarchical decompositions of the search space have been proposed [Lozano-Perez 81].

The following properties of octrees lead to fast path planning algorithms:

1. Octrees provide a spatially-indexed representation of the world. That is, each region of 3-space is associated with a list of objects within the region. Therefore, the objects at each point or region in space can be very quickly retrieved. This leads to very fast collision detection algorithms. During path planning, if the hypothesized motion of an object is represented as the volume the object would sweep out, potential collisions can very quickly be found.
2. Ideally, the search for a path should be performed in a continuous search space. Of course, this potentially leads to an infinite number of paths to be considered during search. Octrees provide a decomposition of free space into cubes, each of which can be treated as a single symbolic unit (i.e., a node) in a search graph. Links are created between two nodes only if their respective cubes are adjacent. In this way, the infinite search space is converted into a finite one.
3. The hierarchical, multiresolution nature of octrees may be utilized to improve the speed of search algorithms. In one technique, for example, only octants at a low resolution level are represented by nodes in the search graph [Kambhampati & Davis 85]. In this way, the graph represents both low resolution EMPTY leaf nodes and low resolution MIXED nodes. By dealing only with octants at a low resolution level, the graph is much smaller and search proceeds more quickly. Paths within each MIXED node region are found separately.
4. Efficient algorithms exist for converting a polyhedral object described by its surfaces into an octree description [Hong 85].
5. Efficient algorithms exist for incrementally modifying octrees [Hong & Shneier 85b, Weng & Ahuja 85]. These techniques assume that a separate database of objects in the world exists. Each object in this database has an octree representation in the object's coordinate system. Incremental modifications to the world octree then consist of rotating, translating, adding, and deleting the object octrees. Of course, not all changes in the world need be immediately reflected in the structure of the octree. Each addition, deletion, or object transformation can simply be remembered in a separate data structure. Modifications to the world octree would then actually take place only when the path search leads to the relevant regions.
6. Finally, octrees are often useful for tasks other than path planning. Because they offer a useful representation of 3-space, they can serve as the output representation for sensory interpretation algorithms [Hong & Shneier 85a, Connolly 84], they can be used to retrieve objects or object features lying in a given region of space, or to solve the hidden feature problem for verification vision or graphics display [Glassner 84, Meagher 82]. The point is that in a complete robot planning, control, and sensory system, octrees may serve in many different kinds of tasks [Shneier et al. 84]. The effective cost of generating the octrees thus becomes lower when compared to the cost in systems that must generate a different description for each task.

The primary disadvantages of octrees are, first, that they do not provide an exact representation of objects and, second, that they tend to require a lot of memory. The first disadvantage can be overcome by using an object's surface-based description when highly precise motions near objects are required. Since the nodes of the octree have pointers to objects contained in them, retrieving the objects in a given region of space is very fast.

The second disadvantage is more difficult to overcome, but techniques such as dynamically expanding the octree into higher resolution levels only when needed, or compressing the octree representation [Gargantini 82], may help. Although spatial decompositions that are irregular [Reddy & Rubin 78, Lozano-Perez 81] may result in smaller trees, operations on them such as locating volume elements, finding their positions, performing translation and rotation, and generating them from surface-based object descriptions are usually much slower.

5. NBS Path Planning System

The remainder of this paper describes a fast, three-dimensional, path planning system under development at NBS. Our goal is to apply the system to a large class of robots, including mobile robots (land, air, and underwater), manipulators with both prismatic and rotary joints, and mobile robots carrying manipulators.

The current implementation of the path planner assumes that the robot's external world is static and structured. We plan eventually to extend the system to unstructured and dynamic worlds by incorporating sensory processing components.

The inputs to the path planner are (1) a description of the robot, (2) a description of the robot's external world, in the form of a single world octree, (3) the configurations of the robot in the start and goal states. The output of the path planner is a piecewise linear path in 3-space. Although pure translation is currently assumed, methods for incorporating rotation will be included in our discussion.

The path generated is always guaranteed to be collision-free, although it is generally not the shortest such path between the start and goal states. Finding the shortest path requires an expensive search. Fortunately, a "reasonably" short path is adequate for many tasks, and such a path can often be found quite quickly.

6. Search Techniques

Three basic search techniques are combined to perform the search through the octree space. The first is hypothesize and test, and involves hypothesizing a simple path for the robot by generating the volume it would sweep out during the motion. Using an algorithm to be described in a later section, a

collision between the swept-out volume and an object in the octree can very quickly be detected. Two kinds of simple paths have been considered: linear paths (corresponding to simple translations) and circular paths (corresponding to simple rotations).

The second search technique is hill climbing. It uses a cost function whose value at any point in free space is proportional to the Euclidean distance from the point to the goal, and whose value at any point inside or on the surface of an object is infinite. The robot is then always made to move to a neighboring point whose cost is the minimum over all neighboring points.* This search technique is very fast because only information local to each robot position is used in deciding in which direction to proceed. The technique is similar to the potential field technique mentioned earlier, and suffers from the same major problem: the algorithm can easily get stuck at a local minimum in the cost function, that is, a point that has a lower cost than any of its neighboring points.

The third technique we use is A^* search [Nilsson 71] -- a best-first, tree-structured search method. The technique is applied to a graph representation of the octree search space, and it performs a global search through the graph. A search tree is built up as the search progresses so that the algorithm can always proceed with the path with lowest cost. Portions of many different paths may therefore be explored before a solution path is finally found. A^* search is therefore more computationally expensive (on average) than hill climbing. However, the minimal cost path in the search graph is always found if $\hat{h}$, the heuristic value of the cost function at any given point, is always less than the actual mimimal cost path from the point to the goal. A search algorithm with this property is said to be *admissible* [Nilsson 71]. In the current implementation, the value of $\hat{h}$ at a point is the Euclidean distance from the point to the goal.

In the future, we intend to use a fourth technique in performing the search, involving a multiresolution search strategy. With this technique, the search is actually performed at relatively low resolution levels in the octree (refer to the work of [Kambhampati & Davis 85] described earlier). Many MIXED nodes are treated as leaves during the search. Paths within each of these node regions are found separately.

The three techniques described above are combined in an attempt to achieve the greatest speed in finding free paths in a variety of world configurations. The search is performed on a graph obtained by connecting the centers of all adjacent EMPTY leaf octants in the octree. Fig. 1 shows the appearance of such a graph for a quadtree example (also see [Kambhampati & Davis 85]). The graph is not explicitly created before the search begins; it is implicitly formed as needed during the search process.

The basic operation of the combined search algorithm is as follows. Beginning at the start state, hill climbing search is performed. If a local miminum is reached, A^* search is invoked, beginning at the point at which hill climbing got stuck (see Fig. 1). The purpose of the A^* search mode is to get out of the valley around this local minimum and over a peak or ridge. At this point, hill climbing may be reinvoked because the robot cannot return to the position at which the local minimum occurred. The process of switching between the two search modes continues until either the goal is reached, or

*The term "hill climbing" in this case is deceiving. Perhaps a better term is "valley descending."

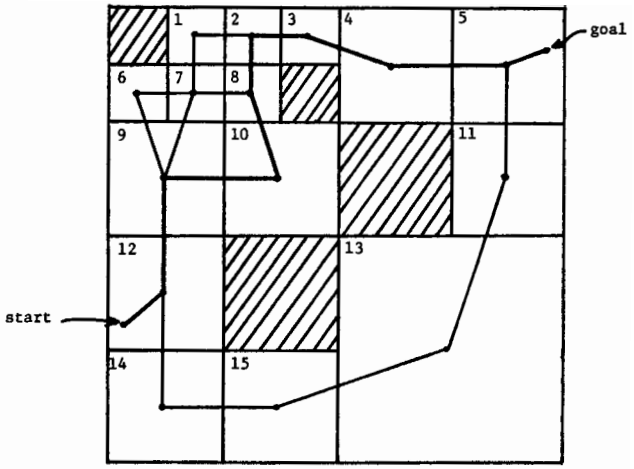

Figure 1: Search graph in which hill climbing and A^* searches are performed. The start and goal positions are shown in blocks 12 and 5, respectively. Hill climbing initially finds a path through blocks 12, 9, and 10. A^* is then invoked and finds a path through blocks 10, 8, and 2. Hill climbing is then reinvoked and finds a path through blocks 2, 3, 4, and 5. As indicated in the text, a swept-volume path will actually be generated from block 2 to the goal, immediately finding the solution path. In this figure, however, we have shown how the search would proceed from block 2 without this.

it is determined that no path to the goal exists. The two search modes provide hypothesis paths for the hypothesize and test technique described earlier. This is done in two ways. First, in determining whether or not a path from one node to an adjacent node in the graph is valid, a linear swept volume for the robot is generated between the two node positions. If no collision with an obstacle is detected, the path is valid. Second, at every node reached during the search, a linear swept volume to the goal is generated. If this volume does not intersect an obstacle, a solution is obtained. Otherwise, the search proceeds as described above.

In general, this algorithm will not result in a minimum cost path from the start to the goal node. The hill climbing mode may lead the robot away from such a path. However, the algorithm will always result in a solution path if one exists (in the search graph). There may be solution paths outside the search graph which, of course, the algorithm will not find. To see why, consider the following. Let d_i be the cost (during hill climbing search) of node n_i in the graph. During hill climbing, either the goal will be reached or a local minimum will be reached. For example, suppose node n_i is reached during hill climbing. Three conditions may occur at this node. If d_i equals the global minimum, then the goal has been reached. If, for each neighbor n_j, $d_i \leq d_j$, then a local minimum has been reached. If there is a neighbor n_k such that $d_i > d_k$ and d_k is a minimum over all neighbors of n_i, then the algorithm will proceed to n_k. Suppose now that A^* is invoked at a local minimum node n_i. Then if it were allowed to run to completion, it would always reach the goal if a path to the goal exists. This is because A^* is admissible, as described earlier. However, if it first reaches an intermediate node n_j such that $d_j < d_i$, hill climbing search will be reinvoked. During hill climbing, the algorithm can never visit node n_i again, since $d_i > d_j$. Therefore, the algorithm will always result in a solution path if one exists.

7. Obtaining Successor Nodes During Search

As described earlier, the search graph is implicitly formed as needed during the search. This occurs by dynamically finding the neighbors, or *successors* [Nilsson 71], of a node in the graph when it is visited. Let n be the current node visited, s_i a possible successor node, and $d(m)$ the cost at any node m (for hill climbing). To obtain a successor node during hill climbing mode, the following steps are taken.

1. Obtain all s_i such that
   (a) s_i is at a neighboring point of n,
   (b) s_i is in an EMPTY octant,
   (c) $d(s_i) < d(n)$, and
   (d) $d(s_i)$ has not yet been computed during the current invocation of hill climbing. (If it has already been computed, then $d(s_i)$ must be greater then $d(n)$. Otherwise, the search would not have reached n.)
2. If the set $\{s_i\}$ = NULL, a local minimum has been reached at node n, so exit.
3. Set p to the potential successor s_j with minimum d.
4. If there is a linear free path from n to p, p is the successor node.
5. Otherwise, remove p from $\{s_i\}$, and go to step 2.

During A^* search mode, there will, in general, be more than one successor for each node. A search tree is generated during the search wherein the successors of each node form the children in the tree. The node s_i is a successor of node n if

1. s_i is at a neighboring point of n,
2. s_i is in an EMPTY octant,
3. successors of s_i have not yet been obtained during the current invocation of A^*,**
4. there is a linear free path from n to s_i.

8. Primitive Shapes

Thus far, we have discussed how the objects in the robot's world are represented in the form of a single world octree. In order to detect potential collisions, the swept volume representing the robot's path must also be represented. One technique used previously is to represent both the swept volume and the world as octrees [Ahuja et al. 80]. Collision detection then consists of traversing the two octrees in parallel. It is assumed here that both octrees are expressed in the same coordinate system. If a node in one tree is FULL while the corresponding node in the other tree is MIXED or FULL, an intersection exists (also see [Boaz & Roach 85] . The problem with this approach is that a complete octree must be generated for each hypothesized swept-volume path. The approach that we use compares the swept volume itself, rather than an octree of the swept volume, to the world octree.

In our approach, the articulate parts of the robot, as well as the swept-volume paths of the robot, are approximated by a set of primitive shapes. Because the robot is always fully contained in its approximating shapes, a free path for the shapes is always a free path for the robot. The converse is not true.

There are three requirements for defining a primitive shape. The first is that computing whether or not the shape intersects an object in the octree should be fast. Our primitive shapes are therefore defined in terms of a *spine* -- either a simple curve or simple surface segment -- and a *radius* -- a single extent outward from the spine that defines the shape's surface. By representing octants in the octree as spheres, the intersection test merely involves determining the shortest distance from the center of a sphere to the spine of the primitive shape, and checking whether or not this distance exceeds the sum of the radii of the sphere and shape. More of this will be described later.

The second requirement for a primitive shape is that it should be a reasonable approximation to a part of the robot or a swept volume. The third requirement is that generating primitive shapes used to represent swept-volume paths should be very fast. This is because the particular shape must be dynamically generated during search. Many of the shapes are therefore defined as translational or rotational sweeps of some other primitive shape.

Some primitive shapes we have considered are the following:

1. Sphere (Fig. 2a). Defined by a center point and a radius.
2. Cylsphere (Fig. 2b). The volume swept out by linear translation of a sphere. Defined by the radius of the sphere and the two end points of the line segment forming the spine.
3. Translation-swept cylsphere (Fig. 2c). The volume swept out by linear translation of a cylsphere. Defined by the radius of the cylsphere and the four end points of the parallelogram forming the spine.
4. Rotation-swept cylsphere. The volume swept out by rotation of a cylsphere about an axis intersecting and perpendicular to its spine. There are two types. For type 1 (Fig. 2d), the rotation angle is less than 180 degrees; for type 2 (Fig. 2e), the angle is greater than 180 degrees. Type 1 is defined by the radius of the cylsphere and the two wedge slices meeting at their apexes that form the spine. Type 2 is defined by the radius of the cylsphere and the pie segment that forms the spine.
5. Torus section (Fig. 2f). The volume swept out by rotation of a primitive shape about an axis that does not intersect the shape.

Fig. 3 indicates how the hand of an IBM 7565 robot might be represented by primitive shapes. Notice that the cables connected to the hand must also be contained in the primitive shapes. Swept-volume paths of the robot might then be formed as follows. To perform some translation of the robot hand, the free space required is a set of translation-swept cylspheres generated for each cylsphere in the description. To perform a given rotation about axis B, the free space required for PART 1 is in the shape of a cylsphere, and for PARTs 2, 3, and 4, torus sections. To perform a given rotation about axis C, the free space required for PART 2 is a rotation-swept cylsphere of either type 1 or 2, and for PARTs 3 and 4, torus sections.

9. Collision Detection

The following algorithm is used to determine whether or not a primitive shape intersects any obstacles represented in the world octree. First, associated with each swept-volume primitive shape is a set of curves within its volume that follow the sweep used to form the shape. If the shape is formed by

**Since the *consistency assumption* for $\hat{h}$ is satisfied [Nilsson 71], a node need never be reexamined once its successors have been obtained.

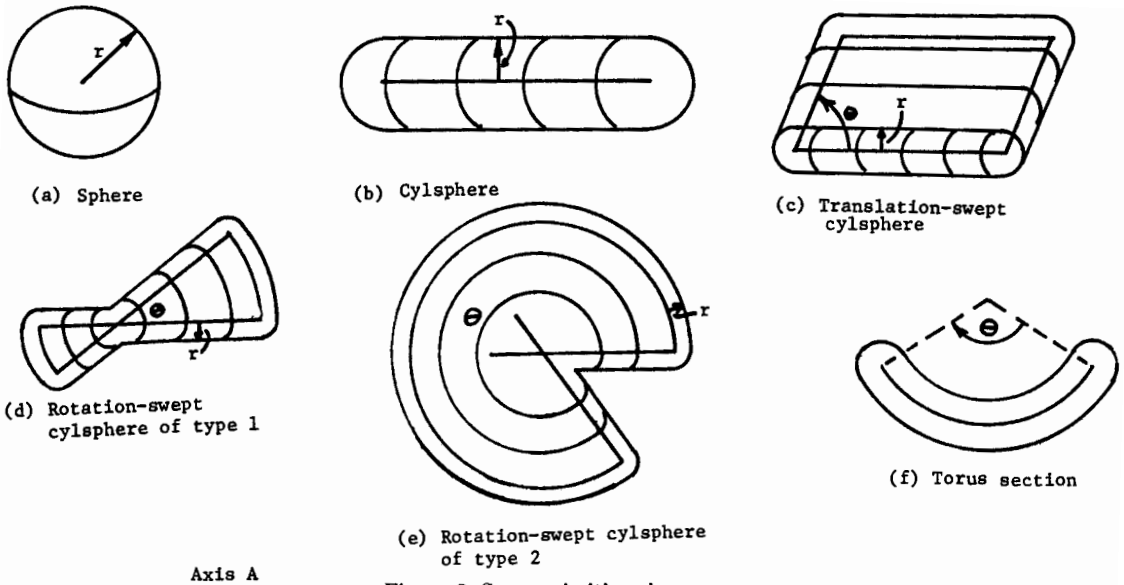

Figure 2: Some primitive shapes.

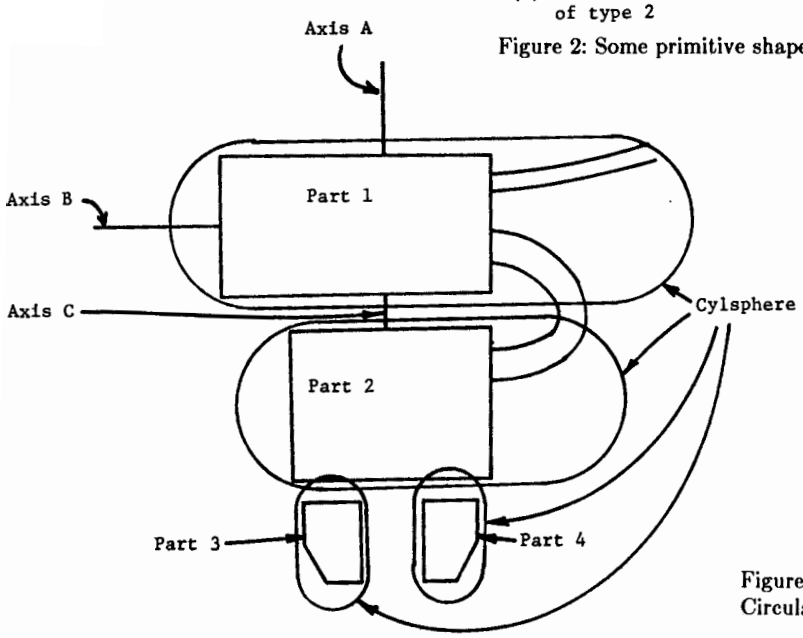

Figure 3: Representing the hand of the IBM 7565 robot. Each articulate part is represented as a primitive shape. Rotations of the hand may occur about axes A, B, or C.

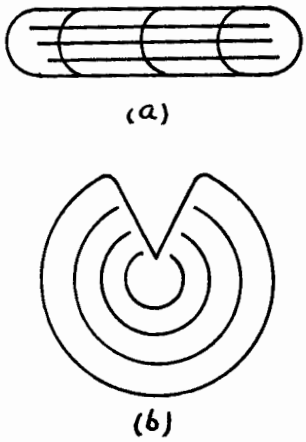

Figure 4: (a) Straight-line segments inside volume. (b) Circular arc segments inside volume.

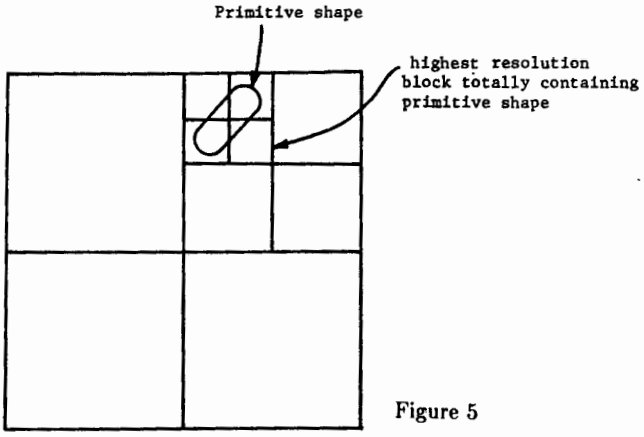

Figure 5

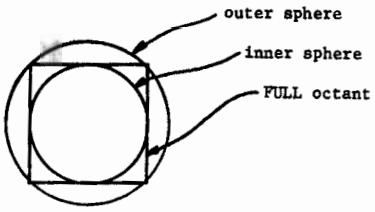

Figure 6

translational sweep, the curves are straight line segments (Fig. 4a). If the shape is formed by rotational sweep, the curves are circular arc segments (Fig. 4b). The curves within the volume of the shape are individually tested to see if any intersects an obstacle. This test is extremely fast [Glassner 84]. If there is an intersection, then of course the shape also intersects the obstacle. If there is no intersection, a more detailed test must be performed on the shape, for some other part of the shape may still intersect an obstacle.

The more detailed test involves performing a breadth-first traversal of the octree, and checking for an intersection between each FULL node and the primitive shape. Using a breadth-first, rather than depth-first, traversal insures that if there is a FULL node at a low resolution level that intersects the shape, it will be found quickly, before much of the rest of the tree is examined.

To avoid visiting and checking for intersection with too many unnecessary nodes in the octree, the highest resolution octant totally containing the bounding box of the primitive shape is initially found (Fig. 5). The breadth-first traversal then occurs within the subtree rooted at this octant. In addition, the children of a MIXED node are not visited unless the node overlaps the bounding box of the primitive shape.

When a FULL node is reached during the traversal, the following tests are performed.

1. If the octant and bounding box of the primitive shape do not overlap, there is no intersection.
2. Let us define the *outer sphere* as the smallest sphere that completely contains the octant, and the *inner sphere* as the largest sphere contained completely within the octant (Fig. 6). If the primitive shape intersects the inner sphere, there is an intersection with the octant [Hong & Shneier 85b].
3. If the shape does not intersect the outer sphere, there is no intersection with the octant.
4. If the shape intersects the outer but not the inner sphere, then
   (a) if the octant is at the highest resolution level, assume an intersection,
   (b) otherwise, divide the octant into 8 suboctants, and for each suboctant, proceed from step 1.

10. Conclusion

This paper has discussed several issues related to fast, 3-D, collision-free motion planning. The NBS path planning system, which is currently under development, has been introduced. The interesting aspects of the system are that (1) path planning occurs in Cartesian 3-space represented as an octree, (2) a hybrid search algorithm is used that combines hypothesize and test, hill climbing, and A^*, and (3) primitive shapes are used to approximate the robot and its swept-volume paths so as to result in fast collision detection algorithms.

As described thus far, the search algorithm does not explicitly handle rotations. We plan to incorporate rotations by decoupling them from translations. That is, the final motion of the robot would involve a mixed sequence of pure translations and pure rotations. This is a common technique used to overcome the huge computational cost associated with searching in a complete high-dimensional space [Brooks 83, Lozano-Perez 81].

We also plan to test our algorithms on several robots, including the IBM 7565 robot. Our aim is to develop a system that works in a variety of real-world situations.

Acknowledgements

Tsai Hong and Mike Shneier have provided excellent ideas and criticism throughout the course of this work. In addition, they read and commented on an earlier version of this paper. Valuable comments were also provided by Ted Hopp, Ernie Kent, and Ron Lumia.

References

1. Ahuja, N., Chien, R.T., Yen, R., and Bridwell, N. "Interference detection and collision avoidance among three dimensional objects." *Proc. First Annual National Conf. on Artificial Intelligence,* Stanford University, August 1980, 44-48.
2. Baer, A., Eastman, C., and Henrion, M. "Geometric modelling: a survey." *Computer-Aided Design,* 11, 1979, 253-272.
3. Besl, P.J. and Jain, R.C. "Three-dimensional object recognition." *Computing Surveys,* Vol. 17, No. 1, 1985, 75-145.
4. Boaz, M. and Roach, J. "An oct-tree representation for three-dimensional motion and collision detection." *SIAM Conf. on Geometric Modeling and Robotics,* Albany, New York, July 1985.
5. Brady, M., Hollerbach, J.M., Johnson, T.L., Lozano-Perez, T., and Mason, M.T., eds. *Robot Motion: Planning and Control.* MIT Press, Cambridge, 1982.
6. Brooks, R.A. "Symbolic reasoning among 3-D models and 2-D images." *Artificial Intelligence,* 17, 1981, 285-348.
7. Brooks, R.A. "Solving the find-path problem by good representation of free space." *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. SMC-13, No. 3, 1983, 190-197.
8. Buckley, C.E. and Leifer, L.J. "A proximity metric for continuum path planning." *Proc. Ninth International Joint Conf. on Artificial Intelligence,* Los Angeles, CA, August 1985, 1096-1102.
9. Canny, J. "A Voronoi method for the piano-movers problem." *IEEE International Conf. on Robotics and Automation,* St. Louis, Missouri, March 1985, 530-535.
10. Connolly, C.I. "Cumulative generation of octree models from range data." *Proc. IEEE International Conf. on Robotics,* Atlanta, GA, March 1984, 25-32.

11. Faverjon, B. "Obstacle avoidance using an octree in the configuration space of a manipulator." *Proc. IEEE International Conf. on Robotics,* Atlanta, GA, March 1984, 504-512.

12. Gargantini, I. "Linear octtrees for fast processing of three-dimensional objects." *Computer Graphics and Image Processing,* 20, 1982, 365-374.

13. Glassner, A.S. "Space subdivision for fast ray tracing." *IEEE Computer Graphics and Applications 4* 10, October 1984, 15-22.

14. Herman, M. "Generating Detailed Scene Descriptions from Range Images." *Proc. 1985 IEEE International Conf. on Robotics and Automation,* St. Louis, Missouri, March 1985, 426-431.

15. Hong, T.-H. Personal communication, 1985.

16. Hong, T.-H. and Shneier, M.O. "Incrementally constructing a spatial representation using a moving camera." *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* San Francisco, CA, June 1985a, 591-596.

17. Hong, T.-H. and Shneier, M.O. "Rotation and translation of objects represented by octrees." Robot Systems Division, National Bureau of Standards, Gaithersburg, MD, October 1985b.

18. Jackins, C.L. and Tanimoto, S.L. "Oct-trees and their use in representing three-dimensional objects." *Computer Graphics and Image Processing 14,* 1980, 249-270.

19. Kambhampati, S. and Davis, L. S. "Multiresolution path planning for mobile robots." Technical Report 127, Center for Automation Research, University of Maryland, College Park, MD, May 1985.

20. Kent, E.W., Shneier, M.O., and Lumia, R. "PIPE (Pipelined Image Processing Engine)." *J. Parallel and Distributed Computing,* 2, 1985, 50-78.

21. Khatib, O. "Real-time obstacle avoidance for manipulators and mobile robots." *IEEE International Conf. on Robotics and Automation,* St. Louis, Missouri, March 1985, 500-505.

22. Lozano-Perez, T. "Automatic planning of manipulator transfer movements." *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. SMC-11, 1981, 681-698.

23. Meagher, D. "Geometric modeling using octree encoding." *Computer Graphics and Image Processing,* 19, 1982, 129-147.

24. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

25. Reddy, D.R. and Rubin, S. "Representation of three-dimensional objects." Technical Report CMU-CS-78-113, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1978.

26. Requicha, A.A.G. "Representations for rigid solids: theory, methods, and systems." *Computing Surveys,* 12(4), 1980, 437-464.

27. Ruff, R. and Ahuja, N. "Path planning in a three dimensional environment." *Proc. Seventh International Conf. on Pattern Recognition,* Montreal, Canada, July 1984, 188-191.

28. Shneier, M., Kent, E., and Mansbach, P. "Representing workspace and model knowledge for a robot with mobile sensors." *Proc. Seventh International Conf. on Pattern Recognition,* Montreal, Canada, July 1984, 199-202.

29. Srihari, S.N. "Representation of three-dimensional digital images." *Computing Surveys,* Vol. 13, No. 4, 1981, 399-424.

30. Thorpe, C. E. "FIDO: Vision and navigation for a robot rover." Technical Report CMU-CS-84-168, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, December 1984.

31. Weng, J. and Ahuja, N. "Octree representation of objects in arbitrary motion." *Proc. IEEE Cobference on Computer Vision and Pattern Recognition,* San Francisco, CA, June 1985, 524-529.